Twin Cubes: Executable Storyboards and Physical Manipulatives for Category Theory / Homotopy Type Theory (CT / HoTT) – including source code

Douglas C. Youvan

doug@youvan.com

August 31, 2025

Twin Cubes turns categorical and HoTT reasoning into pictures you can run—and objects you can hold. Two synchronized, mirrored cubes are driven by a tiny JSON script in a lightweight Python viewer; the same run exports a single PNG storyboard that lists one state per meaningful step. The idea is a visual language: instead of long derivations, readers see arrows act as a sequence of clear, comparable states packaged as a portable "proof card" that drops cleanly into papers, slides, and repositories. The pipeline is intentionally minimal (one Python file + JSON + launcher), so results are easy to regenerate, audit, and cite. The virtual tool is paired with printable left/right cube nets so demonstrations can be reproduced as physical manipulatives; the physical pair serves as a haptic checker and classroom kit. Although cube rotations provide an inviting gateway, the pattern generalizes to small group and category actions on finite displays, supporting identities, functoriality and naturality, unit–counit triangles, coherence/rewriting steps, duals/contravariants, and higher paths. Our aim is an on-ramp: help readers acquire CT/HoTT intuition quickly and share arguments that are both human-readable and machine-reproducible. Source code and example scripts are provided to encourage adoption, remixing, and inclusion of executable supplements alongside traditional expositions.

Keywords: Category Theory, Homotopy Type Theory, visual proofs, executable storyboards, twin cubes, mirrored labeling, rotation group, group actions, display functor, faithfulness, Yoneda, naturality, coherence, adjunctions, diagrammatic reasoning, educational manipulatives, reproducible research, open-source software, printable cube nets. A collaboration with GPT-5. See attachments.

1. Introduction — a visual language you can run

Category Theory (CT) and Homotopy Type Theory (HoTT) offer powerful ways to think, but their *entry cost* is high: proofs are compact yet abstract; diagrams help but are often static; and executable artifacts typically live far from the exposition. Twin Cubes aims to close that gap by turning standard "arrows act on things" reasoning into storyboards you can run and objects you can hold.

Twin Cubes presents two synchronized panels—left-handed and right-handed labelings of the same cube—driven by a short, human-readable script. From the same script, the system renders a live animation and an exported PNG storyboard listing one state per meaningful step (no timing clutter). The storyboard includes a short *proof footer* giving the conventions and the minimal lemma that makes the card stand-alone Q.E.D. The whole pipeline is intentionally small: one Python file, a launcher, and a tiny JSON description of moves. Print-ready left/right cube nets let readers reproduce demonstrations as physical manipulatives.

1.1 Motivation: lower the barrier, keep the rigor

- Fast intuition. Many equalities in CT/HoTT are most convincing when one can *see* two composites act the same way. Storyboards make that action explicit, side-by-side, step-by-step.
- Reproducibility. The artifact you put in a paper is the artifact readers can regenerate. A single script produces both UI and image, so readers can verify and remix without reverse-engineering.
- Physical alignment. Printable nets create a tactile mirror of the virtual run. Physical cubes serve as a *haptic debugger* and a classroom kit, reinforcing the same steps readers see on the storyboard.

1.2 Design choices (readability first)

Twin Cubes adopts diagram mode: face letters remain upright (to reduce cognitive load) while two small cues carry the geometry:

- An orientation cue (e.g., a gold star) always indicates the "upright/top" direction of the *current* front face.

- A distinguished element (e.g., a blue dot) is transported by the scripted moves and is shown on proof steps (rotations, explicit reposition, identity).

With letters upright, readers can follow labels instantly; the cues encode the changing orientation and the evolution of a chosen element without visual noise. The left panel uses the mirror labeling (swapping L/R), the right panel the standard labeling; both execute the same move sequence, inviting comparisons at a glance.

1.3 What the artifact is (and is not)

- Is: a compact *executable storyboard* generator. It renders a single PNG that matches the live run *step-for-step*, with a brief proof footer summarizing conventions and the lemma (e.g., a faithfulness statement) that closes the argument.
- Is not: a general 3D engine, a proof assistant, or a diagram editor. The emphasis is on *clear, reproducible sequences* for small, finite displays and standard identities.

1.4 Minimal workflow

1. Write (or copy) a tiny JSON describing a sequence of moves (quarter-turns about named axes), optional captions, and optional repositioning of the distinguished element.
2. Run the single Python file with the JSON.
3. Read the live panels or open the exported storyboard PNG written next to the JSON.
4. (Optional) Print the left/right nets and repeat the steps with physical cubes.

Because the UI and PNG are generated from the same deduplicated state sequence, what readers see live is exactly what appears on the page. Caption-only and timing “holds” never create spurious duplicate rows; explicit Identity steps are supported when you want a meaningful “no-change” row.

1.5 Why two panels?

The mirrored pair makes contrasts obvious:

- Mirror vs. standard labelings reveal left/right conventions and dualities without extra symbols.
- Before/after comparisons (or “Route A vs. Route B”) land on visibly equal states when the equality holds, which is the essence of many CT/HoTT arguments.
- The pattern generalizes to other paired views: covariant vs. contravariant, primal vs. dual, algebra vs. coalgebra—while keeping the same simple scripting model.

1.6 Scope and generality

Cubes and rotations are a friendly gateway, but the approach is broader: any small group or category action on a finite display set can be scripted. That covers a wide span of classroom and research-adjacent arguments—identities, naturality squares, unit–counit triangles, coherence/rewriting steps, duals/contravariants, and higher paths. The same “storyboard as proof card” idea applies: define what your display functor FFF shows, state (or cite) why it’s separating/faithful enough, and let the visible equality of images carry the conclusion.

1.7 Contributions

- A visual language for CT/HoTT: executable storyboards that make “arrows acting on things” concrete and comparable.
- A single-file toolchain (source code provided) that outputs both a live run and a print-ready PNG with a proof footer, ensuring reproducibility and easy inclusion in papers and slide decks.
- A physical companion: printable, mirrored cube nets aligned with the virtual semantics for hands-on verification and teaching.
- A template for extension: a simple scripting model and rendering pipeline that generalize beyond rotations to other small actions.

1.8 Reader’s map

The next section presents one running example entirely through the storyboard image (no viewer screenshots), explains how to read it, and shows how the footer turns the card into a self-contained Q.E.D. Subsequent sections describe semantics, implementation, reproducibility, physical nets, and generalization, followed by limitations, future work, and references; full source, JSON mini-spec, additional storyboards, and nets appear in the Appendices.

2. Running example — one page, storyboard

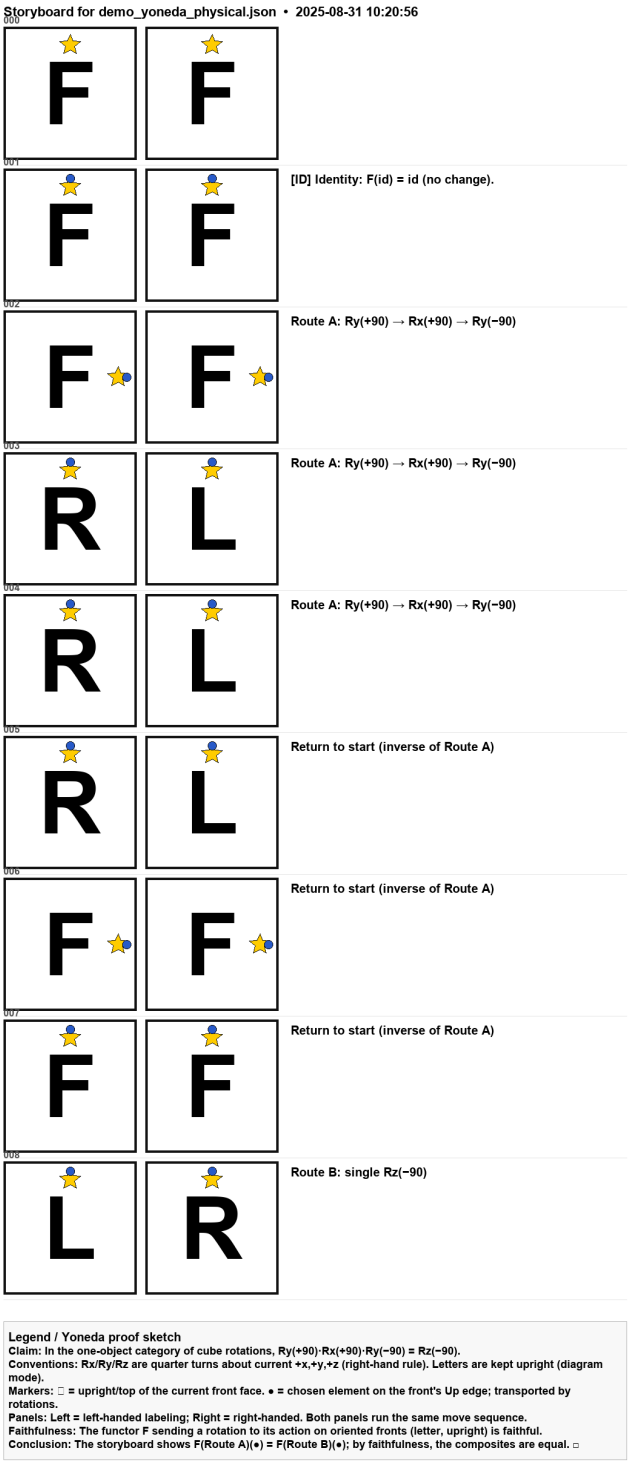

Storyboard for demo_yoneda_physical.json • 2025-08-31 10:20:56
000
001
[ID] Identity: F(id) = id (no change).
002
Route A: Ry(+90) → Rx(+90) → Ry(−90)
003
Route A: Ry(+90) → Rx(+90) → Ry(−90)
004
Route A: Ry(+90) → Rx(+90) → Ry(−90)
005
Return to start (inverse of Route A)
006
Return to start (inverse of Route A)
007
Return to start (inverse of Route A)
008
Route B: single Rz(−90)
Legend / Yoneda proof sketch
Claim: In the one-object category of cube rotations, Ry(+90)·Rx(+90)·Ry(−90) = Rz(−90).
Conventions: Rx/Ry/Rz are quarter turns about current +x,+y,+z (right-hand rule). Letters are kept upright (diagram mode).
Markers: □ = upright/top of the current front face. ● = chosen element on the front's Up edge; transported by rotations.
Panels: Left = left-handed labeling; Right = right-handed. Both panels run the same move sequence.
Faithfulness: The functor F sending a rotation to its action on oriented fronts (letter, upright) is faithful.
Conclusion: The storyboard shows F(Route A)(●) = F(Route B)(●); by faithfulness, the composites are equal. □

Figure 1 (storyboard proof card). Place the exported PNG here. It shows one row per meaningful state, with a brief proof footer at the bottom. No viewer screenshot is used in the paper.

2.1 Claim shown on the card

We exhibit the standard conjugation identity in the rotation category:
Ry(+90) · Rx(+90) · Ry(−90) = Rz(−90).

The storyboard compares two routes (“Route A” vs. “Route B”) on the same starting pose and shows they land in the same oriented front state.

2.2 Conventions used by the card

- Axes are current body axes; signs follow the right-hand rule.
  Rx = quarter turn about +x, Ry about +y, Rz about +z; (+1) means +90°, (−1) means −90°.
- Letters on the front face are kept upright (diagram mode).
- A small gold star marks the “upright/top” direction of the *current* front face.
- A blue dot is a distinguished element transported by moves; it is rendered on proof steps (rotations, explicit reposition, identity) and omitted on pure caption/hold rows to avoid clutter.
- Two panels run the same script: left = mirror (L/R swapped), right = standard labeling.

2.3 How to read the rows

- Each numbered row shows: left panel (mirror labeling), right panel (standard labeling), and an optional caption.
- If two consecutive geometric states would be identical (e.g., after a Hold or caption), they are collapsed; an explicit Identity step is kept as a meaningful no-change row.
- Equality of the two routes is visible when their final rows match (same front letter, same star edge, same dot edge) in both panels.

2.4 Minimal script behind Figure 1 (Listing 1)

This is the entire JSON that produces the card (inline, editable by readers):

```
{
  "title": "Conjugation demo — storyboard proof card",
  "start": "F^U",
  "speed_ms": 350,
  "moves": [
    { "op": "Caption", "text": "Pick x on the Up edge of the front." },
    { "op": "Marker",  "edge": "U" },

    { "op": "Caption", "text": "[ID] Identity: F(id) = id." },

    { "op": "Caption", "text": "Route A: Ry(+90) → Rx(+90) → Ry(−90)" },
    { "op": "Ry", "dir":  1, "steps": 1 },
    { "op": "Rx", "dir":  1, "steps": 1 },
    { "op": "Ry", "dir": -1, "steps": 1 },

    { "op": "Caption", "text": "Return to start (inverse of Route A)" },
    { "op": "Ry", "dir":  1, "steps": 1 },
    { "op": "Rx", "dir": -1, "steps": 1 },
    { "op": "Ry", "dir": -1, "steps": 1 },

    { "op": "Caption", "text": "Route B: Rz(−90)" },
```

```
  { "op": "Rz", "dir": -1, "steps": 1 },


  { "op": "Caption", "text": "Both routes act the same on the oriented front — equality witnessed." }
 ]
}
```

Readers can paste this into a file and regenerate Figure 1 with the provided Python player and launcher. The storyboard produced by the tool exactly matches the deduplicated step sequence used in the live run.

2.5 Why the card is Q.E.D. (the footer, in words)

The card encodes a simple display functor F that records oriented fronts as (front letter, upright edge). An oriented front determines the body frame, so F is faithful: if two rotations agree on one oriented front, they are equal. The storyboard shows F(Route A)(x) = F(Route B)(x) for the chosen element x; therefore Route A = Route B. The footer on the PNG states this claim, the conventions, and the faithfulness note so the card stands alone.

2.6 Variations readers might try

- Start the distinguished element on a different edge (Marker: R/D/L) and regenerate the card.
- Swap the order of turns to test non-equal composites (the final rows will differ).
- Add an explicit Identity step where you want a no-change row to appear in the proof narrative.

3. Design principles and semantics

This section fixes the meanings of states, moves, and what a storyboard asserts.

3.1 Category, objects, and states

- We work in the one-object category Rot whose morphisms are rigid rotations of a cube (composition = do one turn after another; identity = no turn).
- A state in the player is the oriented body frame (r, f, u):
    - f = the cube's current +Y axis (the "front" normal),
    - u = the current +Z axis (the cube's own "up"),
    - r = the current +X axis.
      These are unit, pairwise orthogonal, right-handed (invariant: r × f = u).

3.2 Display functor (what the card shows)

- The storyboard is the action of a display functor F: Rot -> Set.
- F(*) = set of oriented fronts, encoded on the card as a pair:
    1. the front letter (which face is front), and
    2. the upright edge of that face (shown by a small orientation cue).
- Faithfulness note. An oriented front (front letter, upright edge) determines the full body frame (r,f,u). Therefore F is faithful: if two rotations act the same on one oriented front, they are equal. This is the lemma cited in the card's proof footer.

3.3 Move semantics (how JSON composes)

- JSON moves are applied in order (top to bottom).
- Rx, Ry, Rz = quarter turns about the current +x, +y, +z axes of the cube; signs follow the right-hand rule:
    - dir = +1 = +90 degrees, dir = −1 = −90 degrees;

    - steps = k repeats the quarter turn k times.
- Composition convention in prose: “A then B” means do A, then do B (the same order as listed in JSON).

3.4 Diagram-mode visual cues (why letters stay upright)

- Large face letters are drawn upright to reduce cognitive load; we separate orientation from labeling.
- The orientation cue (a small star) marks where the top of the current front face would be if drawn physically. Formally, we map the state’s u vector to one of the four screen edges {U,R,D,L} by comparing it with fixed screen up/right axes and choosing the dominant component.
- A distinguished element (a small dot) represents an element $x \in F(\text{front})$. It is transported by moves and rendered on proof steps (rotations, explicit reposition, identity).

3.5 Token transport (distinguished element)

- At the start, the token lies on a chosen material edge of the current front (default: Up edge).
- Under any rotation, the token’s direction vector is rotated by the same map as (r,f,u), so it arrives at the corresponding edge of whichever face becomes front.
- The token may be repositioned by a Marker step (edge $\in$ {U,R,D,L}); it is never toggled off in this paper’s semantics.

3.6 Duplicate suppression and Identity rows

- The UI and storyboard list one row per distinct state (front letter + orientation edge + token edge). Caption-only and pause steps do not create extra rows.
- When you intend to assert identity (e.g., “apply id”, “no change here”), include a tagged caption like [ID] ... or an explicit Id op; that no-change row is then kept.

3.7 Reading equality on the card

- To show two composites A and B are equal, the storyboard presents two routes that start together and end in the same oriented front (same front letter, same orientation edge) with the same token image.
- By the faithfulness note in 3.2, this equality of images implies A = B in Rot. The footer states the claim, conventions, and the lemma so the card is self-contained.

3.8 Two panels: mirrored labeling

- The left panel uses a mirrored (left/right swapped) labeling; the right panel uses the standard labeling. Both panels execute the same rotation sequence. This makes contrasts (mirror/dual/covariant vs. contravariant) visually immediate without adding notation.

3.9 Invariants and sanity checks

- Always check r, f, u remain orthonormal and right-handed after each move.
- The orientation cue is computed from the state's u (never from screen tricks), so it encodes the cube's actual "top" relative to the viewer.
- The storyboard and the UI are generated from the same deduplicated sequence, guaranteeing step-for-step agreement between what readers see on screen and on the page.

4. System overview

Twin Cubes is intentionally small so the artifact you read is the artifact you can run and share.

4.1 What ships

- Player: one Python file (two_cube_player.py).
- Launcher: a short script (e.g., Launch-2Cube.ps1 on Windows).

- Demo script: a tiny JSON describing moves and captions.
- Outputs: a single PNG storyboard written next to the JSON; it lists one row per meaningful state and includes a brief proof footer.
- (Appendix D) Printable left/right cube nets (for physical manipulatives).

4.2 How it renders (one pipeline for UI and PNG)

- The player parses the JSON into a state sequence by applying moves (Rx/Ry/Rz) and lightweight steps (Caption, Id, Marker to reposition the token).
- The sequence is deduplicated so “holds” and caption-only moments do not create repeated rows. Explicit Identity rows are preserved.
- The same deduplicated sequence drives both the live panels and the exported PNG, ensuring they match step-for-step.

4.3 Visual semantics (always the same)

- Diagram mode: face letters are drawn upright for readability.
- Orientation cue: a small star marks the upright/top of the current front face (derived from the state’s “up” vector).
- Distinguished element: a small dot represents a chosen element; it is transported by moves and shown on proof steps (rotations, explicit reposition, identity).
- Two panels: left uses mirrored labeling; right uses standard labeling. Both run the same move sequence.

4.4 Files in, files out

- Inputs: a JSON file with a title, a start pose (e.g., “F^U”), and a short list of moves.
- Outputs: a PNG named storyboard_<json_basename>.png written in the same folder as the JSON; no other files are required for the paper.

- Encoding: the JSON reader accepts UTF-8 with or without BOM to avoid cross-editor issues.

4.5 Controls (described, not pictured)

- Keyboard: Space = Play/Pause; Left/Right = step; Home/End = first/last.
- Speed control: a small box sets the milliseconds per step when playing.
- No additional UI elements are needed in the paper; we do not reproduce a viewer screenshot.

4.6 Typical workflow for authors and readers

1. Copy the example JSON (Section 2) or write a new one.
2. Run the player with the JSON; the storyboard PNG is generated alongside it.
3. Paste the PNG into the paper as Figure 1.
4. (Optional) Print the left/right nets from Appendix D and physically mirror the steps.

This “single pipeline” design keeps the artifact reproducible: the image readers see is generated by the same code they can execute, and physical nets mirror the same semantics.

5. Reproducibility capsule

This is the one-page, do-this-now guide. Everything is inline text so you can drop it into Word.

5.1 Requirements

- Python: 3.10+ (works on 3.11–3.13).
- Packages: Pillow (for PNG export); tkinter (ships with standard Python on Windows/macOS; on some Linux distros install python3-tk).
- Files you need (from this paper’s artifact):

- two_cube_player.py
- Launch-2Cube.ps1 (Windows convenience; optional on macOS/Linux)
- demo_yoneda_physical.json (the running example)

5.2 Directory (you can choose any folder)

<your-folder>/

two_cube_player.py

Launch-2Cube.ps1

demo_yoneda_physical.json

5.3 Install the one package (if not already present)

Windows / macOS / Linux

python -m pip install --upgrade pip

python -m pip install pillow

(If your system uses python3, substitute python3 for python.)

Linux note: if the UI fails to open, install Tk: e.g., sudo apt-get install python3-tk.

5.4 Run the example

Windows (PowerShell)

PowerShell -ExecutionPolicy Bypass -File "<your-folder>\Launch-2Cube.ps1"

This opens the twin-panels UI and writes a PNG next to the JSON:

<your-folder>\storyboard_demo_yoneda_physical.png

macOS/Linux (Terminal)

cd "<your-folder>"

python two_cube_player.py demo_yoneda_physical.json

Output PNG:

./storyboard_demo_yoneda_physical.png

5.5 What to expect

- A window with two panels (left = mirror labeling, right = standard), step controls, and captions.
- A single PNG storyboard saved alongside the JSON, one row per meaningful state, with a short proof footer.
- The UI and PNG match step-for-step (same sequence, same markers).

5.6 Verify quickly

- Open the PNG and count rows; use the arrow keys in the UI to step through the same count.
- Optional: re-run; the PNG will be regenerated identically (deterministic for the same JSON and environment).

5.7 Customize (first edits to try)

- Open demo_yoneda_physical.json and change the caption text, or switch a turn (e.g., make Rz positive); re-run and observe the last row differ.
- Change the token's starting edge by setting { "op": "Marker", "edge": "R" } near the top.

5.8 Where files go

- PNG is written next to the JSON with the name storyboard_<json-name>.png.
- No cache dirs; no hidden dependencies. JSON is read as UTF-8 (with or without BOM).

5.9 License and citation

- License: *(insert chosen license)*.
- Cite as: “Twin Cubes: executable storyboards and physical manipulatives for CT/HoTT (source code provided)” with a link/DOI to the artifact.

5.10 Physical manipulatives (pointer)

- Printable left/right cube nets are provided in Appendix D. Print at 100%, fold, and trace the same steps by hand to mirror the storyboard.

6. Physical manipulatives

This section tells readers how to build and use the left/right physical cubes so they can mirror the storyboard by hand. The printable nets (left- and right-hand labelings) are provided in Appendix D.

6.1 What you get (Appendix D)

- Two one-page 6-square cross nets:
  – Right-hand cube (standard labeling)
  – Left-hand cube (mirror labeling; R/L swapped)
- Each face square carries its letter and a tiny tick mark on the edge that touches the U face in the net. After folding, that tick mark is the face’s body-up edge—the physical analogue of the storyboard’s “upright star”.

6.2 Print and assemble

- Paper: 65–110 lb cardstock (or ordinary paper if needed).
- Scale: print at 100% (no “fit to page”).
- Cut: along the outer lines. We use no glue tabs—join with small tape on the inside for a clean exterior.
- Fold: crisp folds along all edges; assemble the cross into a cube. Keep letters outside and upright in the *initial* pose (F facing you, U on top).

- Mark the token: place a small removable blue sticker/dot (or pencil dot) on the Up edge of the front face—that is the same initial placement used by the demo JSON.

6.3 Handedness check (quick sanity)

Hold the cube with F facing you and U on top.

- Right-hand (standard) cube: your right hand touches R.
- Left-hand (mirror) cube: your right hand touches L.
  This matches the two virtual panels: left = mirror labeling, right = standard labeling.

6.4 How the physical marks correspond to the storyboard

- The tiny tick marks on each face are fixed to the body. When a face becomes front, the tick may appear on its top/right/bottom/left edge—that is exactly what the storyboard's gold "upright" star denotes. (The storyboard keeps letters upright for readability, but the tick mark tells you how the physical letter would stand.)
- The blue token is a material edge marker. Under Rx/Ry/Rz quarter turns, you carry the token to the corresponding edge of whichever face has swung to the front—precisely what the storyboard's blue dot shows on proof steps.

6.5 Running a demo by hand (example)

- Start pose: F facing, U on top. Blue token on the Up edge of F.
- Route A: Ry(+90) then Rx(+90) then Ry(−90). After each turn, update which face is front and move the token to the carried edge. Note where the face's tick mark appears (top/right/bottom/left).
- Return to start: undo those turns.
- Route B: Rz(−90).

- Compare: Route A and Route B should end in the same front letter, with the face's tick mark on the same edge and the token on the same edge—matching the final storyboard row.

6.6 Classroom & paper-prep tips

- Haptic debugger: have pairs of students run the two routes, then swap cubes (left/right) to see the mirrored labeling in action.
- Reset quickly: a small "home" arrow on the U face helps you re-establish F^U between runs.
- Avoid scaling errors: if physical outcomes disagree with the card, first confirm the print was at 100% and that you used the correct handedness net.
- Photographing figures: a straight-down photo of the two fronts makes a useful companion image in slides (the paper itself uses only the storyboard; see Figure 1).

7. Beyond rotations — pattern and applications

Twin Cubes is a concrete instance of a general recipe: make an action visible, pick a display functor that separates morphisms enough to read equalities off pictures, and export a storyboard that is a portable certificate.

7.1 The pattern (portable to other actions)

Think in four slots:

1. Ambient category or group.
   Examples: the rotation group of a polyhedron; a finite group presented by generators/relations; a small category (objects, arrows).
2. Scriptable generators.
   A tiny alphabet the JSON can name (e.g., r_x, r_y, r_z; or s, t with relations; or f, g, η, ε for adjunctions). Each generator is a state transformer.

3. Display functor F.
   A mapping to a finite “picture set” that the storyboard can render (stickers on faces; ports on a box; corners/edges of a tile; tuples of labels). The goal is faithfulness or at least separation on the equations you care about.

4. Equational goals.
   Two scripts (Route A vs. Route B) that you claim act the same; the storyboard shows F(Route A)(x) = F(Route B)(x) on a chosen probe x, and the footer states the faithfulness/separation fact that closes the argument.

This recipe yields the same authoring experience: tiny JSON in, live panels + PNG out.

7.2 Display choices that travel well

- Diagram mode for labels. Keep glyphs upright (readability); encode orientation with a small cue, as we did with the “upright” marker.
- Distinguished elements. One or more tokens carried by the action (elements, edges, ports).
- Twin (or multi) panels. Use parallel panels for dual/contravariant views, mirror labelings, or before/after states.
- Deduped steps. Always show one row per meaningful change; allow an explicit Identity row when it is the point.

7.3 Ready-to-script application sketches

- Naturality squares. Objects A,B,C,D; arrows f: A→B, g: C→D; a natural transformation α: F⇒G.
  Script two routes around the square (top-right vs. left-bottom) and show the same image of a token at D. Footer cites the naturality condition.
- Adjunction triangles (unit–counit).
  Show that ε_F ∘ Fη = id_F and Gε ∘ η_G = id_G on suitable displays. Storyboards make the “round-trip = identity” visible, with an explicit Identity row.

- Coherence / rewriting.
  Monoidal associativity (α) and unitors (λ, ρ); braid relations; Yang–Baxter; Mac Lane pentagon in miniature. Each equation becomes "Route A vs. Route B end at the same pictured state."
- Commutators and presentations.
  For a finitely presented group ⟨s,t | relations⟩, script the relations (e.g., sts = tst). The storyboard is a finite check of the relator as an equality.
- Duals / contravariants.
  Use the left panel to render the dual or opposite action; the right panel the original. Same script, visually distinct semantics; equalities that should mirror will do so row-for-row.
- Higher paths (HoTT).
  Tokens can carry not just points but small "edge glyphs" to depict paths; parallel panels show homotopies between composites (path concatenation associativity, conjugation). The storyboard records the endpoints' equality while the footer names the homotopy content.

7.4 When faithfulness is too strong

Sometimes F is not globally faithful but still separates the specific equations you care about. State that explicitly in the footer:

- "F distinguishes all composites generated by {…} on probes of the form (…); hence equality on this storyboard implies equality of the scripted composites."

Alternatively, show several probes (e.g., four starting edges) in the same card or as a compact gallery; together they separate the cases you need.

7.5 Authoring checklist for new domains

- Identify generators you can name in JSON and implement as state updates.
- Choose a small, finite picture set for F (what the PNG can render clearly).
- Decide how many panels (mirror, dual, before/after).

- Pick 1–2 tokens that carry the “witness” content.
- State (or cite) the faithfulness/separation fact you rely on; put one sentence in the footer.
- Keep the script short: captions that say what’s being compared, and just enough moves to reach the equality.

7.6 Why this helps

Readers get an audit trail they can replay, print, and even enact physically when appropriate. Authors get portable proof artifacts that survive slides, PDFs, and repositories. And both sides can iterate: edit a JSON line, regenerate, and see whether the equality truly holds.

8. Related work and positioning

Twin Cubes sits at the intersection of diagrammatic reasoning, proof tooling, and hands-on pedagogy. We sketch the landscape and the niche this work occupies.

8.1 Diagrammatic reasoning and visual languages

There is a long tradition of conveying algebraic structure with string diagrams, commuting squares, and rewrite pictures. Many tools let users *draw* such diagrams or *check* local rewrite rules. Twin Cubes complements these by focusing on a single, linear storyboard that witnesses an equality of composites through a finite, reproducible sequence. Rather than a general editor, it is a generator of proof cards: minimal inputs, fixed semantics, deterministic outputs that drop straight into papers.

8.2 Proof assistants and HoTT ecosystems

Modern assistants (e.g., systems supporting dependent types and HoTT) provide machine-checked proofs, formal libraries, and sometimes tactic-driven diagram construction. Twin Cubes is deliberately lighter-weight: it does not certify theorems in a kernel; it exhibits equalities under a small, explicit display functor. The benefit is speed and approachability—authors and readers can iterate on a

JSON script and get an immediately legible artifact—while remaining compatible with formal developments (the same equality can be formalized elsewhere and the storyboard used as a didactic companion).

8.3 Educational manipulatives and visualization

Physical manipulatives have long supported geometric and algebraic intuition; likewise, software visualizations help learners “see” state evolution. Twin Cubes fuses the two: the virtual storyboard and the printable left/right nets are semantics-aligned, so classroom or self-study use can switch between screen and desk without translation. The focus on paired panels (mirror/dual) supports contrastive learning that is harder to achieve with a single view.

8.4 Positioning and contribution

- Artifact, not editor. A tiny pipeline produces a standalone PNG with a short proof footer; readers can regenerate it bit-for-bit from source.
- Executable by design. The same deduplicated step sequence drives the UI and the exported card; there is no divergence between what is seen live and what is published.
- General pattern. Although rotations are the gateway, the recipe extends to small group/category actions and common CT/HoTT identities (naturality, unit–counit, coherence).
- Physical pairing. Printable nets make the argument haptic, an uncommon but valuable complement to screen-only tools.

In short, Twin Cubes prioritizes clarity, reproducibility, and immediacy over breadth of features: it is a compact way to show an equality that readers can run, print, and keep with the paper.

9. Limitations and future work

Twin Cubes is intentionally minimal; that focus brings trade-offs.

Scope of actions. The system targets small group/category actions on finite displays. Richer settings (e.g., large state spaces, parameterized families, continuous rotations) would need either more compact displays or hierarchical "zoom" storyboards.

Not a proof assistant. The storyboard certifies equality *via a stated display functor*. It does not machine-check the lemma (e.g., faithfulness). Future work: optional links to a proof assistant (Lean/Coq/Agda) to export/import equalities or attest the footer lemma.

Diagram mode vs. physical realism. Keeping letters upright maximizes readability but departs from literal stickers. Future option: a toggleable "physical glyph" view, plus a side-by-side comparison for papers that want both.

Tokens and panels. We show one always-present distinguished element and two panels. Extensions: multiple tokens (products, profunctors, parallel witnesses), more panels (dual/contravariant/"before vs. after"), and compact multi-route grids.

Generators and coordinates. We currently use quarter turns about *current* body axes (Rx/Ry/Rz). Future variants: world-axis rotations; custom generators and relations; named composites/macros.

Exports and accessibility. PNG is simple and portable; some venues prefer vector formats. Future: PDF/SVG export; alt-text generation; color-blind-safe palettes; DPI/layout presets; "storyboard hash" for artifact integrity.

Authoring experience. JSON is transparent but spartan. Future: a tiny DSL (still text), inline comments, and a "compare two scripts" mode that auto-lays out Route A vs. Route B.

Empirical evaluation. Classroom studies and usability tests (time-to-comprehension; retention) could quantify learning gains and guide defaults.

10. Conclusion

Twin Cubes offers a compact way to *see* categorical and HoTT equalities: a tiny script produces a live run and a print-ready storyboard that functions as a portable proof card, and printable nets align the virtual semantics with physical manipulatives. The result is an on-ramp: readers acquire intuition quickly, authors share artifacts that are both human-readable and machine-reproducible, and both can iterate by editing a line of JSON and regenerating the card.

Beyond rotations, the same pattern applies wherever a small action can be displayed faithfully (or with sufficient separation): naturality squares, unit–counit triangles, coherence and rewriting identities, dual/contravariant views, and higher paths. Our goal is not to replace formal derivations, but to complement them with *executable storyboards*—pictures you can run, compare, print, and hold. We invite readers to adopt the included source and examples, remix them for their domains, and help shape the next set of extensions (multiple tokens, richer panels, vector exports, and assistant interop).

References

Awodey, S. *Category Theory* (2nd ed.). Oxford University Press, 2010.

Baez, J. C., and Stay, M. “Physics, topology, logic and computation: a Rosetta Stone.” In: *New Structures for Physics*, Springer, 2011.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* Institute for Advanced Study, 2013.

Leinster, T. *Basic Category Theory.* Cambridge University Press, 2014.

Mac Lane, S. *Categories for the Working Mathematician* (2nd ed.). Springer, 1998.

Riehl, E. *Category Theory in Context.* Dover Publications, 2016.

Selinger, P. “A survey of graphical languages for monoidal categories.” In: *New Structures for Physics*, Springer, 2011.

Spivak, D. I. *Category Theory for the Sciences.* MIT Press, 2014.

Warren, M. A. “The homotopy interpretation of type theory.” In: *Handbook of Homotopy Type Theory* (various sources), survey articles.

Wadler, P. “Propositions as Types.” *Communications of the ACM* 58(12), 2015.

Appendices

Appendix A — Reproducibility details

Environment.

- Python: 3.10+ (tested on 3.11–3.13).
- Packages: Pillow (PNG export); tkinter (bundled on Windows/macOS; on some Linux distros install python3-tk).
- OS: Windows 10/11, macOS 12+, Ubuntu 22.04+.

Directory layout (authors/readers may choose any folder).

<project>/

two_cube_player.py

Launch-2Cube.ps1

demo_yoneda_physical.json

storyboard_demo_yoneda_physical.png (generated)

nets_left.png (Appendix D asset)

nets_right.png (Appendix D asset)

Install (once).

python -m pip install --upgrade pip

python -m pip install pillow

Linux UI note: sudo apt-get install python3-tk if Tk is missing.

Run.

- Windows (PowerShell):
- PowerShell -ExecutionPolicy Bypass -File "<project>\Launch-2Cube.ps1"
- macOS/Linux:

- cd "<project>"
- python two_cube_player.py demo_yoneda_physical.json

Outputs.

- A single PNG storyboard written beside the JSON: storyboard_demo_yoneda_physical.png (deterministic for a given JSON + environment).
- Live UI mirrors the storyboard step-for-step.

Fonts & determinism.
The player tries DejaVuSans-Bold, Arial Bold, or system defaults. For bitwise-identical PNGs across machines, install DejaVu Sans and keep the same Pillow version/DPI.

Encoding.
JSON is read as UTF-8 with or without BOM.

Licensing & citation.

- License: *(to be set by authors; e.g., MIT)*
- Cite the artifact as:
  “Twin Cubes: executable storyboards and physical manipulatives for CT/HoTT (source code provided)”.

Integrity check (optional).
Readers may record SHA-256 hashes of two_cube_player.py and the PNG output:

Windows

Get-FileHash two_cube_player.py -Algorithm SHA256

Get-FileHash storyboard_demo_yoneda_physical.png -Algorithm SHA256

macOS/Linux

shasum -a 256 two_cube_player.py

shasum -a 256 storyboard_demo_yoneda_physical.png

Appendix B — Full source code – See attachment to this RG preprint

Listing B.1 — Launch-2Cube.ps1

```
# Launch-2Cube.ps1 — run the twin-cube player from this directory
param(
  [string]$Python = "py",
  [string]$Demo = "demo_yoneda_physical.json"
)

$Dir     = $PSScriptRoot
$player  = Join-Path $Dir "two_cube_player.py"
$demo    = Join-Path $Dir $Demo

if (-not (Test-Path $player)) { Write-Host "two_cube_player.py not found in $Dir" -
ForegroundColor Yellow; exit 1 }
if (-not (Test-Path $demo))   { Write-Host "$Demo not found in $Dir"            -
ForegroundColor Yellow; exit 1 }

Push-Location $Dir
try {
  & $Python -3.13 -u $player $demo
} catch {
  & $Python -u $player $demo
}
Pop-Location
```

Listing B.2 — demo_yoneda_physical.json

```
{
  "title": "Conjugation demo — storyboard proof card",
  "start": "F^U",
  "speed_ms": 350,
  "moves": [
    { "op": "Caption", "text": "Pick x on the Up edge of the front." },
    { "op": "Marker",  "edge": "U" },

    { "op": "Caption", "text": "[ID] Identity: F(id) = id." },

    { "op": "Caption", "text": "Route A: Ry(+90) → Rx(+90) → Ry(−90)" },
    { "op": "Ry", "dir":  1, "steps": 1 },
    { "op": "Rx", "dir":  1, "steps": 1 },
    { "op": "Ry", "dir": -1, "steps": 1 },

    { "op": "Caption", "text": "Return to start (inverse of Route A)" },
    { "op": "Ry", "dir":  1, "steps": 1 },
    { "op": "Rx", "dir": -1, "steps": 1 },
    { "op": "Ry", "dir": -1, "steps": 1 },

    { "op": "Caption", "text": "Route B: Rz(−90)" },
    { "op": "Rz", "dir": -1, "steps": 1 },
```

{ "op": "Caption", "text": "Both routes act the same on the oriented front — equality witnessed." }

]

}

Listing B.3 — two_cube_player.py

(Provided in full in the next section to keep pagination tidy.)

The player is a single file that:
(i) parses the JSON; (ii) produces a deduplicated step sequence;
(iii) renders synchronized left/right panels; (iv) exports a PNG storyboard with a short proof-footer.

Appendix C — JSON mini-spec (authoring guide)

Top-level keys.

- title (string): shown in the window title and PNG header.
- start (string): e.g., "F^U" for “front face F, up face U”.
- speed_ms (int): playback speed; does not affect PNG.
- moves (array): ordered steps.

Moves (in order of frequency).

- {"op":"Ry","dir":±1,"steps":k} — quarter turns about current +y; dir=+1 is +90°, -1 is −90°; steps repeats. Similarly Rx, Rz.
- {"op":"Caption","text":"..."} — explanatory text; caption-only rows are merged onto the next meaningful state.
- {"op":"Id","text":"..."} — explicit identity row (geometry unchanged, kept as a meaningful step).

(Alternatively, use a caption beginning with "[ID] …" to mark the next row as identity.)

- {"op":"Marker","edge":"U|R|D|L"} — (re)position the distinguished element on the current front's edge. The token is never hidden by command.

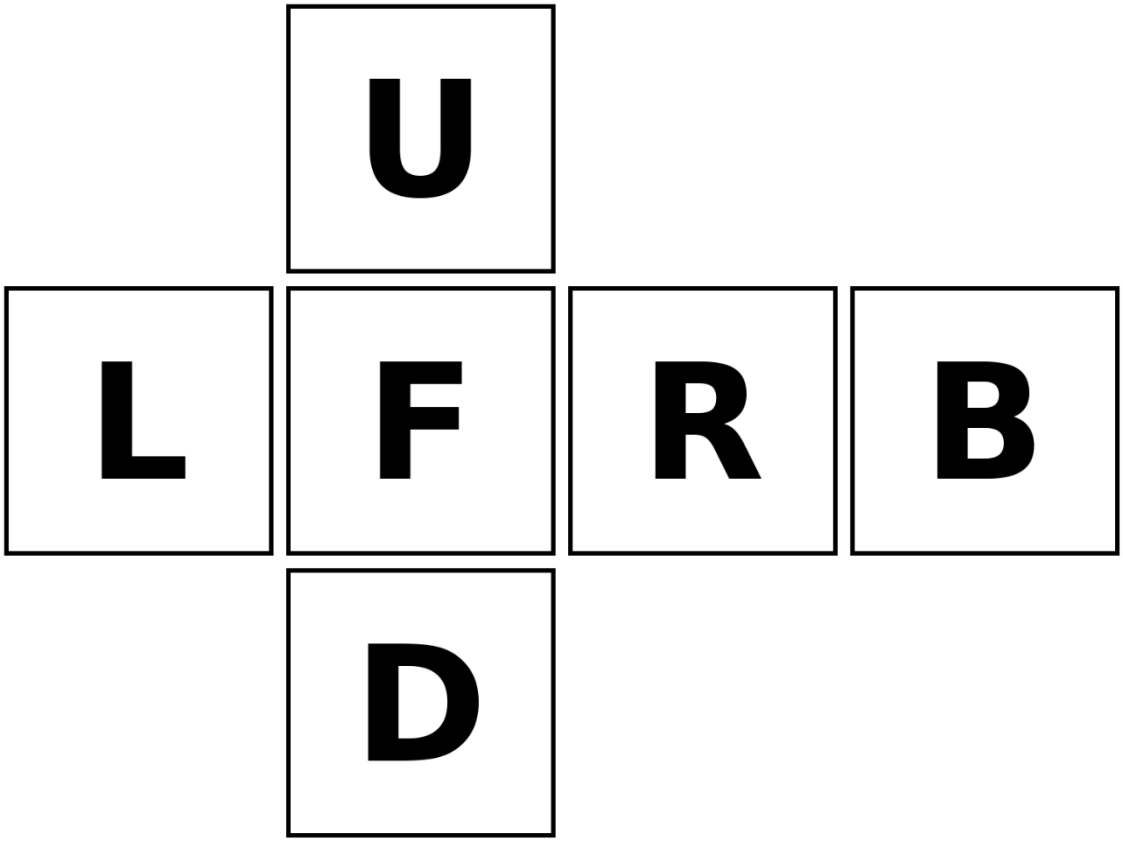
Semantics.

- Letters stay upright (diagram mode).
- Orientation cue is derived from the state's "up" vector and mapped to screen edges U/R/D/L.
- The distinguished element is transported by rotations and shown on proof rows (rot, marker, id).
- Consecutive duplicate geometric states are collapsed, except explicit identity.

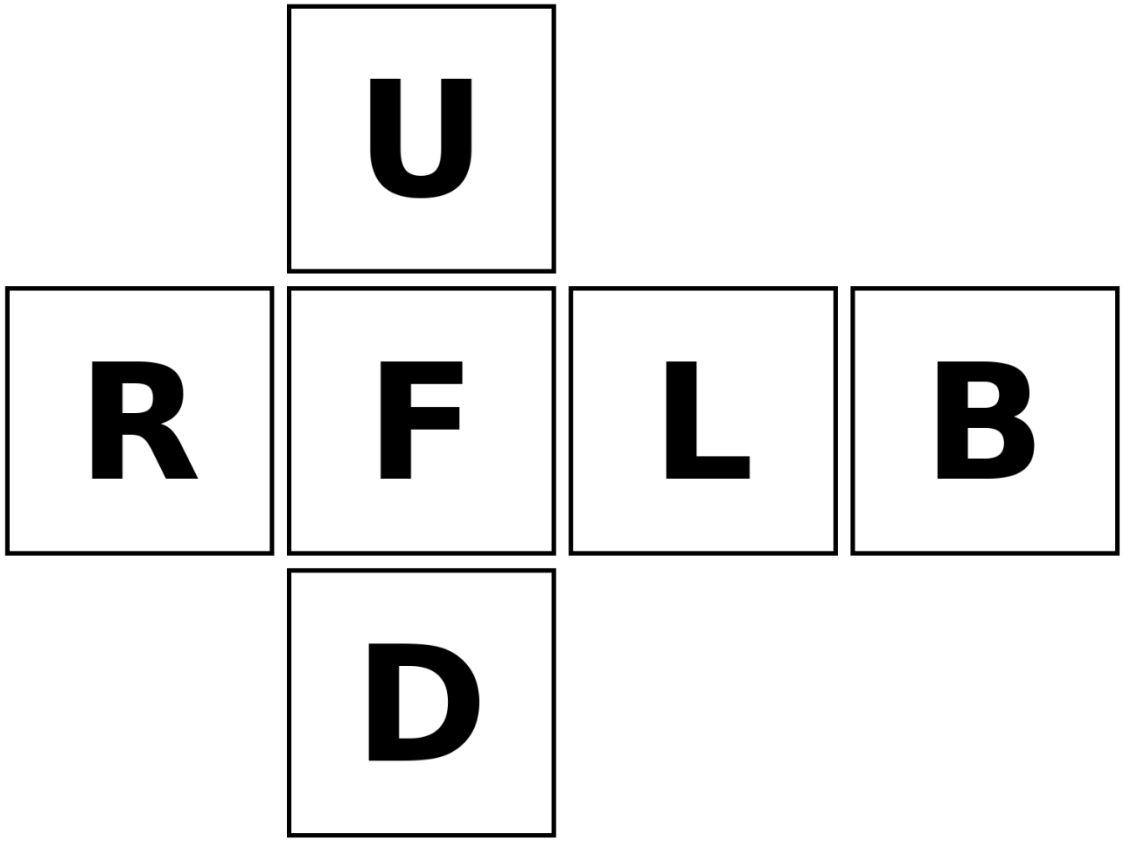
Good practice.

- Keep scripts short (10–20 lines) with one claim per card.
- Use captions to mark "Route A", "Return", "Route B", and the observation line.
- For non-equalities, let the final rows *differ*—the card is still informative.

Appendix D — Printable nets (left/right)



- D.1: Right-hand (standard labeling) net — 6-square cross, letters upright on the sheet; a small tick indicates the edge that will be “body-up” after folding.



- D.2: Left-hand (mirror labeling) net — same cross with L/R swapped.

Assembly notes.
Cut along outer lines; score and fold; tape on the inside (no tabs); start pose F facing, U on top; place a removable blue dot on the Up edge of F to match the demo.

Appendix E — Storyboard gallery (optional)

- E.1: Conjugation demo (Figure 1 of main text).
- E.2: A naturality square example (two routes around a square, equal endpoint).
- E.3: An adjunction triangle identity (round-trip = id, with an explicit identity row).
- E.4: A commutator/presentation fragment (equality fails: final rows differ, shown deliberately).

Each PNG includes a one-line footer stating conventions and the lemma invoked.

Appendix F — Proof notes (faithfulness & sign conventions)

F.1 Faithfulness lemma (or separation condition).
Let F(*) be the set of oriented fronts rendered on the card: pairs (front letter, upright edge). Each such pair determines the full body frame (r,f,u) by setting f to the front's normal and u to the edge-indicated up; then r = u × f. Hence if two rotations agree on one oriented front, they agree on the induced body frame and are equal as rotations. This justifies reading equality off a single probe; we state this in the card's footer.

F.2 Sign conventions.

- Axes are current body axes; composition is "do A, then B" in the listed order.
- Right-hand rule: dir=+1 is +90°, dir=−1 is −90°.
- Conjugation example: Ry(+90)·Rx(+90)·Ry(−90) equals Rz(−90) under these conventions (a one-line matrix check can be added if desired).

Appendix G — Troubleshooting & tips

- PNG missing: check Pillow is installed (pip install pillow).
- UI won't open (Linux): install Tk (sudo apt-get install python3-tk).
- Weird characters / JSON error: ensure UTF-8 encoding; the player accepts BOM and non-BOM.
- Fonts differ across machines: install DejaVu Sans Bold for stable layout.
- No storyboard written: confirm you passed a JSON path and have write permissions to the folder.
- Physical vs. virtual mismatch: confirm you used the correct handedness net and printed at 100%.
- Need vector graphics: export PNG at high DPI or post-process with a vectorization tool; native PDF/SVG export is a plausible extension.